

LISTÁK

Listák 2-2

Lista: fej (hd), fark (tl)

- A nem-üres lista első eleme a lista *feje*.

```
(* hd : 'a list -> 'a
   hd xs = a nem-üres xs első eleme (az xs feje)
*)
fun hd (x :: _) = x;
```

- A nem-üres lista első utáni elemeiből áll a lista *farka*.

```
(* tl : 'a list -> 'a list
   tl xs = a nem-üres xs első utáni elemeinek az eredetivel
          azonos sorrendű listája (az xs farka)
*)
fun tl (_ :: xs) = xs;
```

- `hd` és `tl` *parciális* függvények. Ha könyvtárbeli megfelelőiket (`List.hd`, `List.tl`) üres listára alkalmazzuk, `Empty` néven *kivételt* jeleznek.
- `_` az ún. *mindenesjel*: mindenre illeszkedő minta. Kifejezésben – pl. az egyenlőségjel jobb oldalán – nem használható.

Listák: hossz (length), elemek összege (isum), szorzata (rprod)

- Egy lista hosszát adja eredményül a length függvény (vö. List.length).

```
(* length : 'a list -> int
   length zs = a zs elemeinek száma *)
fun length []          = 0
  | length (_ :: zs) = 1 + length zs;
```

- Egy egész számokból álló lista elemeinek összegét adja eredményül isum.

```
(* isum : int list -> int
   isum ns = az ns egészlista elemeinek összege *)
fun isum []           = 0
  | isum (n :: ns) = n + isum ns;
```

- Egy valós számokból álló lista elemeinek szorzatát adja eredményül rprod.

```
(* rprod : real list -> real
   rprod xs = az xs valós lista elemeinek szorzata *)
fun rprod []         = 1.0
  | rprod (x :: xs) = x * rprod xs;
```

Példák: hd, tl, length, isum, rprod

- hd, tl

A kifejezés	Az mosml válasza
List.hd [1, 2, 3];	> val it = 1 : int
List.hd [];	! Uncaught exception: ! Empty
List.tl [1, 2, 3];	> val it = [2, 3] : int list
List.tl [];	! Uncaught exception: ! Empty

- length, isum, rprod

A kifejezés	Az mosml válasza
length [1, 2, 3, 4];	> val it = 4 : int
length [];	> val it = 0 : int
isum [1, 2, 3, 4];	> val it = 10 : int
isum [];	> val it = 0 : int
rprod [1.0, 2.0, 3.0, 4.0];	> val it = 24.0 : real
rprod [];	> val it = 1.0 : real

LOKÁLIS KIFEJEZÉS

Lokális kifejezés

- *Lokális kifejezést* használunk, ha ismétlődő részkifejezéseket *csak egyszer* akarunk kiszámítani, vagy akkor, ha bizonyos értékeket a program többi része elől *el akarunk rejtetni*.

- Szintaxisa:

```
let d          ahol d nemüres deklarációsorozat,  
in e          e nemüres kifejezés.  
end
```

- Példa:

```
(* length : 'a list -> int  
   length zs = a zs elemeinek száma *)  
fun length zs =  
  let  
    (* len : 'a list -> int  
       len zs = a zs elemeinek száma *)  
    fun len [] = 0  
      | len (_ :: zs) = 1 + len zs  
  in  
    len zs  
  end;
```

LISTÁK

Listák: adott függvény alkalmazása lista minden elemére (map)

- Példa: vonjunk négyzetgyököt egy valós számokból álló lista minden eleméből!

```
map Math.sqrt [1.0, 4.0, 9.0, 16.0] = [1.0, 2.0, 3.0, 4.0]
```

- Általában: $\text{map } f [x_1, x_2, \dots, x_n] = [f x_1, f x_2, \dots, f x_n]$

- map definíciója:

```
(* map : ('a -> 'b) -> 'a list -> 'b list
   map f xs = az xs f szerint transzformált elemeiből álló lista
*)
fun map f [] = []
  | map f (x :: xs) = f x :: map f xs;
```

- map típusa:

```
('a -> 'b) -> 'a list -> 'b list = ('a -> 'b) -> ('a list -> 'b list)
```

Ugyanis a \rightarrow jobbra köt!

Azaz ha `map`-et egy `'a -> 'b` típusú függvényre alkalmazzuk, akkor egy `'a list -> 'b list` típusú **függvényt** ad eredményül. E függvényt egy `'a list` típusú listára alkalmazva egy `'b list` típusú listát kapunk.

PROGRAMHELYESSÉG

Programhelyesség 2-10

A program helyességének igazolása a map példáján

- A rekurzív programról be kell látnunk, hogy
 - ◇ funkcionálisan helyes (azt kapjuk eredményül, amit várunk),
 - ◇ a kiértékelése biztosan befejeződik (nem „végtelen” a rekurzió).
- Bizonyítása hossz szerinti *strukturális indukcióval* lehetséges (visszavezethető a teljes indukcióra).

```
fun map f [] = []  
  | map f (x :: xs) = f x :: map f xs;
```

- ◇ Feltesszük, hogy a map jó eredményt ad az eggyel rövidebb listára (azaz a lista farkára). Alkalmazzuk az f-et a lista első elemére (a fejére). A fej transzformálásával kapott eredményt a fark transzformálásával kapott lista elé fűzve valóban a várt eredményt kapjuk.
- ◇ A kiértékelés véges számú lépésben befejeződik, mert a lista véges, a map függvényt a *rekurzív ágban* minden lépésben egyre rövidülő listára alkalmazzuk, és gondoskodtunk a rekurzió leállításáról (a *triviális eset* kezeléséről, ui. van nem rekurzív ág).

LISTÁK

Listák 2-12

Listák: adott predikátumot kielégítő elemek kiválogatása (`filter`)

- Kitérő: `explode`, `implode`

- ◇ `explode` : `string -> char list`
pl. `explode "abc" = ["a", "b", "c"]`
- ◇ `implode` : `char list -> string`
pl. `implode ["a", "b", "c"] = "abc"`

- Példa: gyűjtsük ki a kisbetűket egy karakterlistából!

```
List.filter Char.isLower (explode "VaLtOgAtVa") =  
    ["a", "t", "g", "t", "a"];
```

- Általában, ha

$p\ x_1 = \text{true}, p\ x_2 = \text{false}, p\ x_3 = \text{true}, \dots, p\ x_{2k+1} = \text{true},$

akkor

`filter p [x1, x2, x3, ..., x2k+1] = [x1, x3, ..., x2k+1].`

Lista: filter (folyt.)

- filter definíciója

```
(* filter : ('a -> bool) -> 'a list -> 'a list
   filter p zs = a zs p-t kielégítő elemeiből álló lista
*)
fun filter _ [] = []
  | filter p (x :: xs) =
    if p x then x :: filter p xs else filter p xs;
```

- filter típusa, ha egyargumentumú függvénynek tekintjük (-> jobbra köt!):

```
filter : ('a -> bool) -> ('a list -> 'a list).
```

Azaz ha filter-t egy 'a -> bool típusú függvényre (predikátumra) alkalmazzuk, akkor egy ('a list -> 'a list) függvényt ad eredményül. Ezt egy 'a list típusú listára alkalmazva egy 'a list típusú listát kapunk.